```
; NTFS Bootstrapping Stage 2 , 15 Sektoren nach dem vbr
; Der BIOS lädt den ersten Sektor (VBR) an 0x0000:0x7C00 (physikalisch 0x07C00)
; Der VBR-Code lädt dann weitere 15 Sektoren (insgesamt 8192 Bytes)
; Diese werden direkt hinter dem VBR in den Speicher gelegt



%define TELETYPE_AUSGABE 0x0E

; ntldr segment after load in memory
%define NTLDR_SEGMENT    0xD00
%define NTLDR_ANSPRUNG   0x026A

%define ADR_CHS_SEKTORANZAHL_GESAMT    0x20
%define ADR_CHS_ANZ_GELESENER_SEKT     0x10
%define ADR_CHS_ANZ_RESERVIERTER_SEKT  0x0e
%define LAUFWERKSKENNUNG               0x24
%define ADR_SEKTOREN_PRO_SPUR          0x18
%define ADR_ANZAHL_KOEPFE              0x1a
%define VBR_VERSATZ                    0x1c

%define FLAG_LBA_SUPPORT_INITIALISIERT 0x14
%define VBR_LADEADRESSE                0x7c0
%define STACK_SEGMENT                  0x7C00
%define STACK_OFFSET                   0x0000

    org VBR_LADEADRESSE
;times VBR_LADEADRESSE db 0


vbr_ladeadresse:
    jmp short 0x54             ; Sprung zu start
    nop
    db  'NTFS    '             ; OEM Name (NTFS + padding)

; NTFS BPB und Extended BPB Struktur (aus Bootsektor)
; ---------------------------------------------------
; Offset 0x0B: Bytes Per Sector (WORD)
BytesPerSector     dw    0x0200       ; 512 Bytes pro Sektor

; Offset 0x0D: Sectors Per Cluster (BYTE)
SectorsPerCluster  db    0x08         ; 8 Sektoren pro Cluster

; Offset 0x0E: (WORD)  ADR_CHS_ANZ_RESERVIERTER_SEKT
ReservedSectors    dw    0x0000       ; 0 bei NTFS  , wird auf 0x10 gesetzt.
;                                       After load win2000 into memory , set to how many sectors of
the Boot Record are
;                                       to be read into Memory; it begins with a value of 16
(decimal) and counts down to zero.
;                                       bearbeitet in zeile 0x6C

; Offset 0x10:  4 Bytes  ADR_CHS_ANZ_GELESENER_SEKT
llx10              dd    0x00000000  ; Anzahl der gelesenen Sektoren CHS

; Offset 0x14   FLAG_LBA_SUPPORT_INITIALISIERT
LBA_int            db    0x00    ; ungleich 0x00 bedeutet LBA-Support schon initialisiert

; Offset 0x15: Media Descriptor (BYTE)
MediaDescriptor    db 0xF8         ; Festplatte

; Offset 0x16: immer 0 (WORD)
Padding2           dw 0x0000       ; immer Null

; Offset 0x18: Sectors Per Track (WORD)  ADR_SEKTOREN_PRO_SPUR
SectorsPerTrack    dw 0x003F       ; 63 Sektoren pro Spur (Little Endian)

; Offset 0x1A: Number Of Heads (WORD)  ADR_ANZAHL_KOEPFE
NumberOfHeads      dw 0x00FF       ; 255 Köpfe pro Zylinder

; Offset 0x1C: (DWORD)   VBR_VERSATZ
HiddenSectors      dd 0x0000003F   ; 63 Versteckte Sektoren= MBR vor VBR,  offset für
ADR_CHS_ANZ_GELESENER_SEKT

; Offset 0x20:  (DWORD)
chs_sektorenanzahl:   dd 0x00000000   ; Gesamte CHS-Sektoranzahl

; Offset 0x24: (DWORD)  LAUFWERKSKENNUNG
laufwerk_0024:     db 0x80    ; Laufwerkskennungen 80=lw c:
                   db 0x00, 0x80, 0x 00   ; unbekannter Wert
```

```
; Offset 0x28: Total Sectors (QWORD)
TotalSectors       dq 0x00000000005F99C6  ; Gesamtsektoren (64-Bit) 6 261 830

; Offset 0x30: Logical Cluster Number für $MFT (QWORD)
MFTClusterNumber   dq 0x0000000000000004  ; Startcluster $MFT

; Offset 0x38: Logical Cluster Number für $MFTMirr (QWORD)
MFTMirrClusterNum  dq 0x000000000005F99C  ; Startcluster $MFTSpiegel

; Offset 0x40: Clusters Per File Record Segment (DWORD)
ClustersPerFileRec dd 0x000000F6   ; Cluster pro Dateieintragsegment

; Offset 0x44: Clusters Per Index Block (DWORD)
ClustersPerIndex   dd 0x00000001   ; Cluster pro Indexblock

; Offset 0x48: Volume Serial Number (QWORD)
VolumeSerialNum    dq 0x808CD8E98CD8DB2C  ; Volume Seriennummer

; Offset 0x50: Checksum (DWORD)
Checksum           dd 0x00000000   ; Prüfsumme Bootsektor


; Initialisierung des Stacks und Segmente
; ---------------------------------------
start:  cli                        ; Interrupts deaktivieren (critical section)
      xor ax, ax                   ; AX = 0
      mov ss, ax                   ; Stack Segment (SS) = 0x0000  STACK_OFFSET
      mov sp, STACK_SEGMENT        ; Stack Pointer (SP) = STACK_SEGMENT → Stack wächst nach unten
      sti                          ; Interrupts wieder aktivieren

; Daten-Segment setzen
      mov ax, VBR_LADEADRESSE
      mov ds, ax                   ; VBR_LADEADRESSE ist Data Segment (DS)

; Aufruf CHS Berechnung
      call geometrieinformationen  ; Aufruf: CHS-Berechnung (logische Größe → CHS-Umrechnung)

; ES auf NTLDR_SEGMENT setzen – Ziel für Datentransfer oder temporärer Speicher
      mov ax, NTLDR_SEGMENT        ; AX = NTLDR_SEGMENT,  Speicher-Ziel-Segemnt
      mov es, ax                   ; ES = NTLDR_SEGMENT (es setzen für int13h-42  Segment als Ziel
des Ladens von Disk)

; Register vorbereiten
      xor bx, bx                   ; BX = 0

; Vorbereitung: BIOS-Diskettenparameter oder Flags
      mov byte [ADR_CHS_ANZ_RESERVIERTER_SEKT], 0x10 ; Speicherstelle 0x000E = 0x10 setzen

; Aufruf einer Funktion, vermutlich BIOS-Disk-Lese-Funktion
      call disk_transfer_to_memory ; dap aufbauen, transfer ausführen, fehler melden

; Adresse vorbereiten und kontrollierter Rücksprung mit `retf`
      push word NTLDR_SEGMENT      ; Push ES-Segment (für `retf`)
      push word NTLDR_ANSPRUNG     ; Push Offset → Zieladresse: NTLDR_SEGMENT:NTLDR_ANSPRUNG,
Ansprung NTLDR
      retf                         ; Far-Ret (sprung zu NTLDR_SEGMENT:NTLDR_ANSPRUNG) → Ausführung
fortsetzen



;  Geometrieinformationen des BIOS (über INT 13h, AH=08) und berechnet dann die Anzahl der Sektoren
auf der Festplatte per CHS.
geometrieinformationen:
    mov dl, [LAUFWERKSKENNUNG]        ; DL = BIOS-Laufwerksnummer (z. B. 0x80 = erste Festplatte)
    mov ah, 0x08          ; BIOS-Funktion: Get Drive Parameters
    int 0x13              ; BIOS-Aufruf: Rückgabe u.a. von CH, CL, DH

    jnc .weiter           ; Springe, wenn kein Fehler auftrat

    ; Fehlerfall – BIOS liefert keine Werte, setze CHS auf max:
    mov cx, 0xFFFF        ; CX = 0xFFFF → CL = 0xFF, CH = 0xFF
    mov dh, cl            ; DH = 0xFF → maximale Kopfanzahl (255)

.weiter:
    ; Kopfanzahl berechnen (Head Count = DH + 1)
    movzx eax, dh         ; EAX = DH (zero-extended auf 32 Bit)
    inc ax                ; Head Count = DH + 1 → EAX

    ; Sektoren pro Track (Bits 0–5 von CL)
    movzx edx, cl         ; EDX = CL
```

```
    and dl, 0x3F          ; Maske: Nur untere 6 Bit (Sektoranzahl, 1-basiert)

    mul dx                ; EAX *= DX → EAX = Kopfanzahl × Sektoren/Track

    ; Zylinderanzahl rekonstruieren aus CH und CL (Bits 6–7)
    xchg cl, ch           ; CL = CH, CH = CL (ursprünglich)
    shr ch, 6             ; CH = Bits 6–7 von ursprünglichem CL → High-Zylinder
    inc cx                ; Zylinderanzahl = ((CH << 8) | CL) + 1

    movzx ecx, cx         ; ECX = CX (zero-extended)
    mul ecx               ; EAX *= ECX → Gesamte CHS-Sektoranzahl

    mov [ADR_CHS_SEKTORANZAHL_GESAMT], eax       ; chs_sektorenanzahl, Speichere Ergebnis an
Speicherstelle 0x20
    ret                   ; Rückkehr zur aufrufenden Routine


; Testen ob bios extension int13h ah=0x42 vorhanden
test_int13_extention:
    mov ah, 0x41                                 ; Funktion 0x41: Prüfen, ob BIOS INT 13h
Erweiterungen vorhanden sind
    mov bx, 0x55AA                               ; BX mit Signaturwert 0x55AA vorbereiten (Erwarteter
Rückgabewert)
    mov dl, [LAUFWERKSKENNUNG]                   ; DL = Laufwerksnummer aus Speicheradresse 0x24
    int 0x13                                     ; BIOS-Disk-Interrupt aufrufen
    jc l0886                                     ; Bei Fehler (Carry gesetzt) Fehlerbehandlung bei
0x886
    cmp bx, 0xAA55                               ; BX soll 0xAA55 zurückliefern, sonst keine
Unterstützung (Signatur prüfen)
    jnz l0886                                    ; Springe bei fehlender Signatur zum Fehlerhandler
    test cl, 0x1                                 ; Prüfe, ob Bit 0 in CL gesetzt ist (Indikator für
INT 13h Erweiterung vorhanden)
    jz l0886                                     ; Wenn Bit 0 nicht gesetzt, Fehlerhandler aufrufen
    inc byte [FLAG_LBA_SUPPORT_INITIALISIERT]    ; Intialisierung erfolgt
l0886:   ret                    ; Zurückkehren aus der Funktion

; ------------------------------
; dap aufbauen , transfer von disk in den memory
disk_transfer_to_memory:
          pushad                  ; Alle 32-Bit-Register sichern
          push ds
          push es

disk_lesen:    mov eax, [ADR_CHS_ANZ_GELESENER_SEKT]
          add eax, [VBR_VERSATZ]

          ; Wenn EAX < [ADR_CHS_SEKTORANZAHL_GESAMT] (chs_sektorenanzahl = Endsektor+1), dann
springe zu CHS-Pfad
          cmp eax, [ADR_CHS_SEKTORANZAHL_GESAMT]
          jc near  chs_berechnung     ; CHS-Berechnung verwenden

          ; LBA-Modus wird verwendet
          push ds
          o32 push byte 0x00          ; Assembleranweisung: push dword 0x00000000
          push eax                    ; LBA-Adresse
          push es                     ; Zielsegment
          push bx                     ; Zieloffset (z. B. 0)
          push dword 0x10010          ; 0001 00 10 -> Sektoranzahl reserviert "länge dap"

          ; Prüfen, ob LBA-Support schon initialisiert wurde
          cmp byte [FLAG_LBA_SUPPORT_INITIALISIERT], 0x00
          jnz near start_transfer          ; Überspringe Init, wenn bereits gesetzt
          call test_int13_extention   ; Testen ob bios extension int13h ah=0x42 vorhanden, dl muß lw
beinhalten

          cmp byte [FLAG_LBA_SUPPORT_INITIALISIERT], 0x00
          jz near fehlermeldung_ausgeben   ; Fehler? d.h. Initialiesierung nicht erfolgt,  gehe zu
Fehlerbehandlung

start_transfer:
          ; Transfer von Disk in den Speicher
          mov ah, 0x42            ; BIOS-Funktion 0x42: LBA-Read, al undefiniert
          mov dl, [LAUFWERKSKENNUNG]          ; BIOS-Laufwerksnummer (z. B. 0x80 für HDD)

          push ss                 ; Stack-Segment in DS laden (für Zeigerübergabe)
          pop ds
          mov si, sp              ; DS:SI zeigt auf den Disk-Address-Packet (DAP)
          int 0x13                ; BIOS Sektor lesen (LBA)

          ; Stack wieder aufräumen
```

```
          pop eax
          pop bx
          pop es
          pop eax
          pop eax
          pop ds
          jmp short fuellen_ntfs_parameterblock         ; Weiter zum füllen NTFS-Parameter-Block

; --- CHS-Berechnungs-Pfad ---
; CHS und LBA müssen synchron gehalten werden, unabhängig davin, wie gelesen wurde
chs_berechnung:
          xor edx, edx
          movzx ecx, word [ADR_SEKTOREN_PRO_SPUR]
          div ecx                 ; EAX / ECX → EAX = Zylinder, EDX = Rest (Sektor-1)

          inc dl                  ; BIOS-Sektornummer beginnt bei 1
          mov cl, dl              ; CL = Sektor

          mov edx, eax
          shr edx, 16             ; DL = High-Zylinder
          div word [ADR_ANZAHL_KOEPFE]          ; Division durch Anzahl Köpfe
          xchg dl, dh             ; DH = Kopfnummer

          mov dl, [LAUFWERKSKENNUNG]          ; BIOS-Laufwerksnummer
          mov ch, al              ; Zylinder (low byte)
          shl ah, 6               ; AH = bits 6–7 → für CHS High-Zylinder
          or cl, ah               ; CL = Sektor + obere Zylinderbits

          mov ax, 0x0201          ; int 13h Funktion 02h (lesen), 1 Sektor
          int 0x13                ; BIOS Disk Read

; Aktualisieren NTFS-Parameter-Block
fuellen_ntfs_parameterblock:
          jc near fehlermeldung_ausgeben                ; Fehler → Fehlerbehandlung

          ; ES um +0x20  Segmentregister der Ladeadresse einen Lese-Sektor weiterschalten,
          ; so daß offset unverändert beleibt, Vorbereitung für nächsten Lesevorgang
          mov ax, es
          add ax, 0x20
          mov es, ax

          inc dword [ADR_CHS_ANZ_GELESENER_SEKT]    ; Anzahl der gelesenen Sektoren inkrementieren

          ; 0x6c
          dec word [ADR_CHS_ANZ_RESERVIERTER_SEKT]  ; Anzahl reservierter (geladenen) Sektoren
dekrementieren
          jnz near disk_lesen                           ; Noch nicht fertig → zurück zur Lese-
Schleife

          pop es
          pop ds
          popad
          ret

; --- Fehlerbehandlung oder Abschluss ---
fehlermeldung_ausgeben:
    mov al, [0x1f8]  ; 1. Zeichen aus Fehler-String
    call l0930
    mov al, [0x1fb]         ; 2. Zeichen aus Fehler-String
    call l0930
    sti
    jmp short $             ; Endlosschleife bei Fehler

; --- Zeichen anzeigen ---
l0930:
    mov ah, 0x01            ; Tastatur: Status abfragen? oder Standardwert für `int 10h`
    mov si, ax
.loop:
    lodsb                   ; AL = [DS:SI], SI++
    cmp al, 0x00
    jz .done
    mov ah, TELETYPE_AUSGABE            ; int 10h, Teletype-Ausgabe
    mov bx, 0x0007          ; Seite 0, Attribute = Lichtgrau auf Schwarz
    int 0x10
    jmp short .loop
.done:
    ret

l0183:  db 0x0D, 0x0A                       ; CR LF
l0185:  db "Fehler beim Lesen des Datentr"
```

```
        db 0x84                             ; 'ä'
        db "gers", 0x00
l01a8:  db 0x0D, 0x0A                       ; CR LF
l01aa:  db "NTLDR fehlt", 0x00
l01b6:  db 0x0D, 0x0A                       ; CR LF
l01b8:  db "NTLDR komprimiert", 0x00
l01ca:  db 0x0D, 0x0A                       ; CR LF   098A
l01cc:  db "Neustart mit Strg+Alt+Entf"
l01e6:  db 0x0D, 0x0A                       ; CR LF
        db 0x00                             ; Nullterminator

l01e9:  db 0x0A, 0x00, 0x00, 0x00
l01ed:  db 0x00, 0x00, 0x00, 0x00
l01f1:  db 0x00, 0x00, 0x00, 0x00
l01f5:  db 0x00, 0x00, 0x00

l01f8:  db 0x83 ; "Fehler beim Lesen des Datentr"
l01f9:  db 0xA8 ; "NTLDR fehlt",
l01fa:  db 0xB6 ; "NTLDR komprimiert"
l01fb:  db 0xCA ; "Neustart mit Strg+Alt+Entf"


l01fc:  dw 0x0000                           ; 2 Bytes
; Bootsektor-Signatur
l01fe:  dw 0xAA55                           ; 2 Bytes
```